

CIRCULATION COPY
SUBJECT TO RECALL
IN TWO WEEKS

UCRL- 83910
PREPRINT




The SCALD Timing Verifier:
A New Approach to Timing Constraints
in Large Digital Systems

Thomas M. McWilliams

This paper was prepared for submittal to:
IEEE Transactions on Circuits and Systems

January, 1980



Lawrence
Livermore
Laboratory

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

The SCALD Timing Verifier: A New Approach to Timing Constraints in Large Digital Systems

Thomas M. McWilliams

Lawrence Livermore Laboratory
University of California
and
Computer Science Department
Stanford University

ABSTRACT

A new approach to the verification of the timing constraints on large digital systems has been developed. The algorithm is computationally very efficient and also provides the early and continuous feedback about the timing aspects of synchronous sequential circuits as they are designed. It also allows for the design to be conveniently verified in sections, permitting the verification of designs which would otherwise be too large to do on existing computer systems. A system using this algorithm has been implemented, and has been used to verify the timing constraints on the design of the S-1 Mark IIA processor, which consists of 10,000 ECL chips, and is comparable in performance to the Cray-1 CPU.

INTRODUCTION

The SCALD Timing Verifier takes in the design of a synchronous sequential system given in the SCALD hardware description language, and analyzes the circuit, searching for timing errors. SCALD (Structured Computer Aided Logic Design) is a complete CAD system that inputs a graphics-based, hierarchical description of a design, and generates a complete set of low level documentation and magnetic tapes to implement the design [2,3,4]. The Timing Verifier does a complete timing verification based on the minimum and maximum propagation delays of the circuit components, their set-up and hold times, minimum pulse width constraints, and wire delays.

One of the main features of the SCALD Timing Verifier is the ability to verify designs by modules. This not only permits the use of computers with limited memory size, but also allows timing constraints to be checked as a design progresses, on a day-by-day basis. This is particularly important in that it allows timing errors to be corrected before they have a chance to propagate their effects throughout the design, or to induce major design changes late in the design. It also supports an accurate estimation of the cycle time of a machine before the design is completed.

PREVIOUS APPROACHES

There have been a number of previous approaches to the verification of timing constraints in digital systems; these can be grouped into two main categories: logic simulation [1,5] and worst-case path analysis [6].

The logic simulation approach poses several problems. It requires either a complete design including any microcode and

diagnostics, or some way of generating patterns to drive the undefined signals. Waiting until the design is completed to start simulation presents the problem that bugs are not found until late in the design cycle. Generating patterns to drive undefined signals is very time-consuming and difficult, especially when the patterns need to go through the "worst-case" set of states. Simulation is also a very inefficient way of finding timing errors, because of the need to run through a large number of states in order to test all of the "worst-case" timing paths. In fact, for most large digital systems, it is impossible to have a high degree of confidence that the worst-case states have all been tested, and both the human and computational efforts required to do simulation is orders of magnitude greater than that required by the Timing Verifier.

The worst-case path analysis approach examines all paths through the combinational logic between registers or latches, searching for the longest and shortest paths. This approach only works on fairly simple combinational logic, and tends to fall down on complex synchronous sequential circuits. It is also computationally expensive, but does provide feedback to the designer early during the design cycle, without the need to generate detailed test patterns.

In contrast, the SCALD Timing Verifier eliminates most of the problems associated with these approaches, allowing the design to be verified as it proceeds, without the need to generate complex complex test patterns. It also uses a computationally efficient algorithm to cover all of the states needed to identify and quantitate all timing errors. Handling circuits where logic simulation of parts of the circuit is needed to understand the detailed timing is another of its capabilities.

A NEW APPROACH

A new approach to verifying satisfaction of timing constraints on large digital systems will be described here. This system operates on synchronous sequential systems, and checks all of the logic level timing errors which occur within those system. These include the non-satisfaction of the set-up, hold, or minimum pulse width time requirements for registers, latches, and other complex functions. In addition to these errors, it checks the timing on control signals which are ANDed with clock signals, to verify that they are stable while the clock is asserted, in order to avoid any possible bogus specification of control-conditioned clock lines. The Timing Verifier takes into account both the minimum and maximum propagation delay of

all of the system's components and of the interconnections between them.

Theory of the Timing Verifier

Within synchronous sequential circuits, most signals can only be changing during particular parts of the clock period. For example, it may be possible for a particular signal to be changing only during the second half of the clock cycle, given that all of the components making up the system are within their timing specifications.

Consider a register that can be clocked only at a particular time within the clock period. The output of the register can change only during a short time after it is clocked, so it is guaranteed to be stable for the entire clock period except around the point at which it is clocked. The output of a gate driven from this register can then be changing only during a period of time determined by its propagation delay and when the output of the register is changing.

Determining when a given signal may be changing and when it is stable within the clock period is the key step for the Timing Verifier. Once this has been done, it is relatively easy to check all of the timing constraints placed on the circuit. For instance, in order to check the set-up and hold times on a register, all the Timing Verifier has to do is to see if its input could be changing at a time that it might be clocked.

If the timing of the circuit never depended on the values of signals, but merely on when they were changing or stable, the Timing Verifier would be very simple. Clock signals have a value which is periodic, and have the same value every cycle, so they are easy to handle. The signals which are difficult to treat are those whose values effect the circuit timing, and which have different values during different cycles of the circuit. For example, a control signal which determines whether a register is clocked during a given cycle affects whether the output of the register might change that cycle. If the circuit counts on the register not changing every cycle, then the Timing Verifier must do *case analysis* to keep from generating false error messages. This requires the Timing Verifier to check the type of cycle when the control signal is true, and to check the type of cycle when it is false. This is potentially a very time-consuming process, but it has turned out not to be. This is because most signals have a "worst-case" state. For example, the worst-case for most registers is to assume that they are clocked every cycle. Only in those situations where both the clocked and unclocked cases need to be checked separately does the Timing Verifier have to compute both of them. In those cases, the Timing Verifier remembers the values of all the signals which are not affected by the signal which is having case analysis done on it, and thus has to recompute only the signals which change with the signal being analysed.

For a given circuit, the Timing Verifier has some number of cases to analyze. Which signals require case analysis and what cases need to be evaluated are specified by the designer. It has been found that most circuits have a fairly small number of these cases which need to be analyzed.

The basic procedure for the Timing Verifier is then to take the first case, and to calculate for each signal in the system when it could be changing during the clock cycle. Once it has done this, it checks for possible violation of timing constraints

for that case. It then goes on to the next case to be checked, only recomputing those signals which are different from the first case, and checking for any possible timing errors in that case. Continuing this process, it checks all of the cases, thereby performing a complete check of the circuit for timing constraint violations.

Circuit Period

The circuit being verified must contain one basic clock, whose period is specified to the Timing Verifier. If different parts of the circuit being verified run at different clock rates, then the period specified to the Timing Verifier is the least common multiple of the different clock periods. For example, a processor might have an instruction unit that has a period of 30 nsec and an execution unit that has a period of 15 nsec. In this case, the period specified to the Timing Verifier would be 30 nsec. Clock signals which occur within the circuit may occur at any phase with the basic clock period.

Circuit Model

Circuits are described to the Timing Verifier in terms of gates, registers, latches, set-up and hold constraints, and minimum pulse width constraints. More complex functions are then defined in terms of these primitives, through the use of graphic-based macros, using the SCALD Hardware Description Language [2,3].

The following sections define the values that are used to represent the behavior of signals, and the definitions of the primitive components for these possible values.

Value System Used To Represent Signals. At any instant, each and every signal in the circuit has one of seven values:

Value	Meaning
0	false, or zero
1	true, or one
S or STABLE	signal is stable, i.e., not changing
C or CHANGE	signal may be changing
R or RISE	signal is going from a zero to a one
F or FALL	signal is going from a one to a zero
U or UNKNOWN	initial value used for all signals

Signal Values

The value of a signal the clock period is represented by a linked list, each node of which specifies a value and the duration of that value. The sum of the durations of all the nodes in the list must equal the period of the circuit being analyzed.

When a signal propagates through a gate or wire where it is delayed by a variable amount of time, then skew is added to the signal, representing the uncertainty in when the signal will subsequently change. This skew is maintained separately in the representation of the signal to preserve information about the width of pulses in the signal, in order to avoid bogus timing errors asserting that minimum pulse width requirements

have not been met. If two or more changing signals are combined, the skew then cannot be simply represented separately. It is therefore incorporated into the signal representation by using the CHANGE, RISE, and FALL values.

Definition of Combinational Functions. This section defines the basic combinational functions used by the Timing Verifier. All other combinational functions are then defined in terms of a combination of these definitions.

The following tables define the INCLUSIVE-OR (OR), EXCLUSIVE-OR (XOR), AND, CHANGE (CHG), and NOT functions for the seven value logic system used in the Timing Verifier.

A OR B

		B → 0 1 S C R F U						
A ↓	0	0	1	S	C	R	F	U
	1	1	1	1	1	1	1	1
	S	S	1	S	C	R	F	U
	C	C	1	C	C	C	C	U
	R	R	1	R	C	R	C	U
	F	F	1	F	C	C	F	U
	U	U	1	U	U	U	U	U

Figure 1

Definition of inclusive-or function

A AND B

		B → 0 1 S C R F U						
A ↓	0	0	0	0	0	0	0	0
	1	0	1	S	C	R	F	U
	S	0	S	S	C	R	F	U
	C	0	C	C	C	C	C	U
	R	0	R	R	C	R	C	U
	F	0	F	F	C	C	F	U
	U	0	U	U	U	U	U	U

Figure 2

Definition of and function

A XOR B

		B → 0 1 S C R F U						
A ↓	0	0	1	S	C	R	F	U
	1	1	0	S	C	F	R	U
	S	S	S	S	C	C	C	U
	C	C	C	C	C	C	C	U
	R	R	F	C	C	C	C	U
	F	F	R	C	C	C	C	U
	U	U	U	U	U	U	U	U

Figure 3

Definition of exclusive-or function

A CHG B

		B → 0 1 S C R F U						
A ↓	0	S	S	S	C	C	C	U
	1	S	S	S	C	C	C	U
	S	S	S	S	C	C	C	U
	C	C	C	C	C	C	C	U
	R	C	C	C	C	C	C	U
	F	C	C	C	C	C	C	U
	U	U	U	U	U	U	U	U

Figure 4

Definition of change function

NOT A

A ↓	0	1
	1	0
	S	S
	C	C
	R	F
	F	R
	U	U

Figure 5

Definition of not function

The output of the CHANGE function has the value CHANGE if any of its inputs are changing; otherwise it has the value STABLE. It is a useful function in modeling complex combinational logic, where the actual function being performed is not important to the verification process. Common examples are in the modeling of parity trees and adders, for which the Timing Verifier cares only when the outputs of these circuits are changing, not for their actual value.

Definition of Registers and Latches. The Timing Verifier has two models for registers, which are shown in Figure 6. The first register model just has "CLOCK" and "DATA" inputs, and can only change its output on the rising-edge of its "CLOCK" input. The output of the register will be set to the "CHANGE" state between the time determined by the minimum and maximum delays of the register following the rising-edge of "CLOCK". Unless the "DATA" input is a true or false during the rising-edge of the "CLOCK" input, the output will be set to the "STABLE" value for the rest of the cycle; otherwise, it will be set to the value of the "DATA" input. The example shows a minimum delay of 1.0 nsec and a maximum delay of 3.8 nsec being specified for the register, which is 36-bits wide.

The second register shown in Figure 6 is the same as the first register, except that it has asynchronous "SET" and "RESET" inputs in addition to the "DATA" and "CLOCK" inputs. If either the "SET" or "RESET" inputs are made to be non-zero, then they control the operation of the register and the setting or resetting of its output after the specified propagation delay of the register.

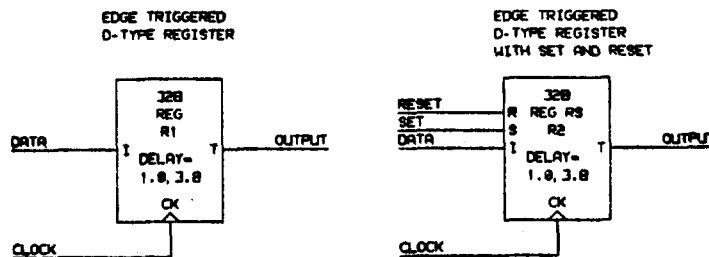


Figure 6

Two register models used by Timing Verifier.

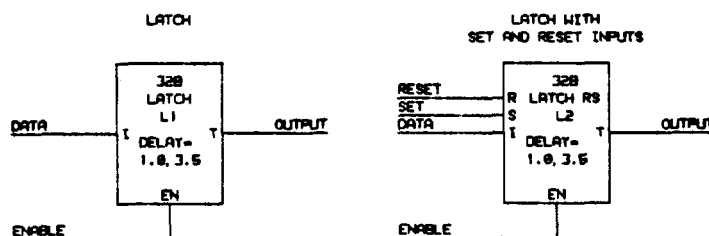


Figure 7

Two latch models used by Timing Verifier.

The Timing Verifier has two models for latches, which are shown in Figure 7. The "OUTPUT" of the first latch merely follows the "DATA" input when the "ENABLE" input is high, and is stable for the remainder of the cycle. The second latch is the same as the first except for the addition of the asynchronous "SET" and "RESET" inputs, which set or clear the latch when the "ENABLE" input is low, after the specified propagation delay.

Assertions on Signals

In order to be able to treat partially designed circuits, the Verifier needs timing assertions on undefined signals. Undefined signals with no assertions are taken to be always stable, to prevent them from giving rise to numerous spurious error messages. Two types of assertions are used for specifying clocks, and one is used for defining the behavior of control and data signals.

There are two categories of clock signals: precision and non-precision. The only difference between precision and non-precision clock specifications is the default skew used when none is explicitly given. Skew is generated by the variation in the wire delay to the different parts of a large digital system and by the variation in delay between the different buffers used in the clock generation. In the design of a large digital system, these variations can become quite large, and may degrade performance unacceptably. To reduce this skew, the shorter clock paths can have additional delay deliberately inserted into them. Because the delays in a clock distribution system may vary between successive implementations of a design, in many cases it must be adjusted by hand, by using some type of adjustable delay for each of the clock lines. Using this technique, the skew can be reduced to below some designer-specified amount. In order to verify the timing in a design which has been so de-skewed, it is necessary to describe how the clocks will be adjusted in detail within the design

specification. A number of features have been provided to make this task as easy as possible, and will be described in this and the next section.

If a clock signal is adjusted to some specified skew, then an assertion can be given within the signal name denoting that fact. Assertions are given at the end of signal names and are preceded by a period. They are considered part of the signal name by the rest of the SCALD system, which thereby guarantees that all of the assertions for a given signal are consistent.

The format for the assertions for the precision and non-precision clocks are:

SIGNAL NAME .P <value specification> <skew specification>

and

SIGNAL NAME .C <value specification> <skew specification>

where:

<value specification> => <time range> |
 <time range> , <value specification>
<time range> => <time> | <time> - <time>
<time> => <real number>
<skew specification> => | (<minus skew> , <plus skew>)
<minus skew> => <negative real or zero>

An example clock specification is

XYZ .C4-6 L

which says that the signal goes from high to low at time 4, and

from low to high at time 6. The time units in which the clocks are specified are normally some fraction of the cycle time. For example, one eighth of the cycle is the basic clock interval used in the design of the S-1 Mark IIA processor. Specifying clock intervals as fractions of the cycle time (rather than in absolute time units) allows the relative timing within the design to be scaled automatically if the cycle time is changed. The signal

XYZ.C2-3,5-6

is high from 2 to 3 and from 5 to 6, and is low for the rest of the clock cycle. If a single time is given instead of a range, then a time interval of one clock unit is assumed. For example,

XYZ.C2,5

is equivalent to the previous signal.

The type of assertion applicable to the behavior of control and data signals specifies when a given signal is stable, and when it may be changing. Its general form is

SIGNAL NAME S <value specification>

For example, the name XYZ.S4-8 says that the signal is stable from time 4 to time 8, and may be changing during the rest of the cycle.

This type of assertion has several uses. First, it allows the designer to specify his assumptions about when signals are valid (i.e., not changing) as he creates them in the design process, and those assumptions will be used by the Verifier until the signals are generated by hardware designed subsequently. For signals so generated, the designers initial assertion is checked against the timing of the actual generated signal, and an error message is outputted if the assertion is thereby violated. In the design of the S-1 Mark IIA processor, most signal names have stable assertions in them. This greatly improves the readability of the logic, since a signal name very explicitly includes a specification of when the associated signal is valid.

Putting the stable assertion on interface signals is the key to the ability to verify a design in sections. After each section is verified, SCALD checks to see that the interface signals have the same timing assertions. If no section has an error, and all of the interface signals have the same assertions on them, then the entire design must be free of timing errors.

Evaluation Directives

Evaluation directives tell the Timing Verifier how to evaluate certain gates. They can also specify the exact point in a circuit at which a clock is adjusted to some specified time.

Because the Verifier is not doing full logic simulation on a completed design, it does not know the logic level of most signals, but only whether the signals are stable or changing. Consider the circuit shown in Figure 8. The clock signal "CK.P2-3 L" is being ANDed to the control signal "WRITE.S0-6 L" to generate a write enable pulse for the RAM array. If the data is stable every cycle during the period that the RAM is to be written, then the most efficient way to check for timing errors is just to analyze the case in which the signal "WRITE.S0-6 L" enables a write. The "&H" directive shown at the end of the clock signal says to ignore the value of the "WRITE

.S0-6 L" signal, allowing the clock signal always to propagate through the gate. In addition, it says the timing specified by the clock signal is to be adjusted so that it refers to the time at which the output, rather than the input, of the gate changes. The "&H" directive also specifies to check that the control signal "WRITE.S0-6 L" is stable during the period that the clock is asserted, to ensure that the write will be either solidly enabled or solidly disabled.

There are a number of different directives of the general type of the "&H" directive. For example, the "&Z" directive on the signal "CK.P0-4" states that the clock timing refers to the time at which the output of the gate changes. If multiple directives are given after a signal, such as "&HZ", then the first letter refers to the first level of gating and the second refers to the second level of gating after the directive. There is no limit on the length of a directive string.

DESIGN SPECIFICATION

Figure 8 shows a circuit example specified in the SCALD Hardware Description Language. The circuit consists of a 16-word by 32-bit RAM, a 32-bit register, a 2-input multiplexer and several gates. This design description is entered into the computer via an interactive graphic editor, and forms the data base for driving the entire SCALD system.

A detailed description of the basic SCALD language can be found in [2,3,4] and will not be repeated here. The main points of interest to the Timing Verifier in Figure 3 are the assertions on signals, evaluation directives, and the specification of possible wire delays. The assertion on the signal "W DATA.S0-6<0.31>" says that it is stable from time 0 to time 6, allowing the Verifier to check the timing of this circuit without knowing how the signal is generated. The assertion on the clock signal "CK.P2-3 L" says that it is low between times 2 and 3, and high for the rest of the cycle. The signal "ADR<0.3> [0.0,6.0]" states that the 4 address wires on the RAM can be between 0.0 and 6.0 nsec long. The evaluation directives "&H" and "&Z" have already been described.

CHIP DEFINITIONS

For each chip used in a design, a definition of its timing and logical properties are given in the SCALD Hardware Description Language.

A chip is defined in terms of a set of primitive functions which the Verifier understands. These primitives include AND, OR, XOR, NOT, and CHANGE gates, registers, latches, multiplexers, a set-up and hold checker, and a minimum pulse width checker. Two typical chip definitions are shown in Figures 9 and 10. Figure 9 shows the definition of a 10145A, a 16-word RAM. Figure 10 shows the definition of a 10158, a 2-input multiplexer. The 10145A model is a timing model only. The "3 CHG" and "4 CHG" gates are CHANGE gates, which output the value CHANGE when any of their inputs change, and output the value STABLE the rest of the time. Using CHANGE gates has been found to be invaluable in modeling complex functions for which knowledge of the exact logical operation is unnecessary. The model for the 10158, on the other hand, is an accurate model, which could be used to do full logic simulation. For the 10158, the model of its complete logical operation is necessary to verify timing constraints in many circuits.

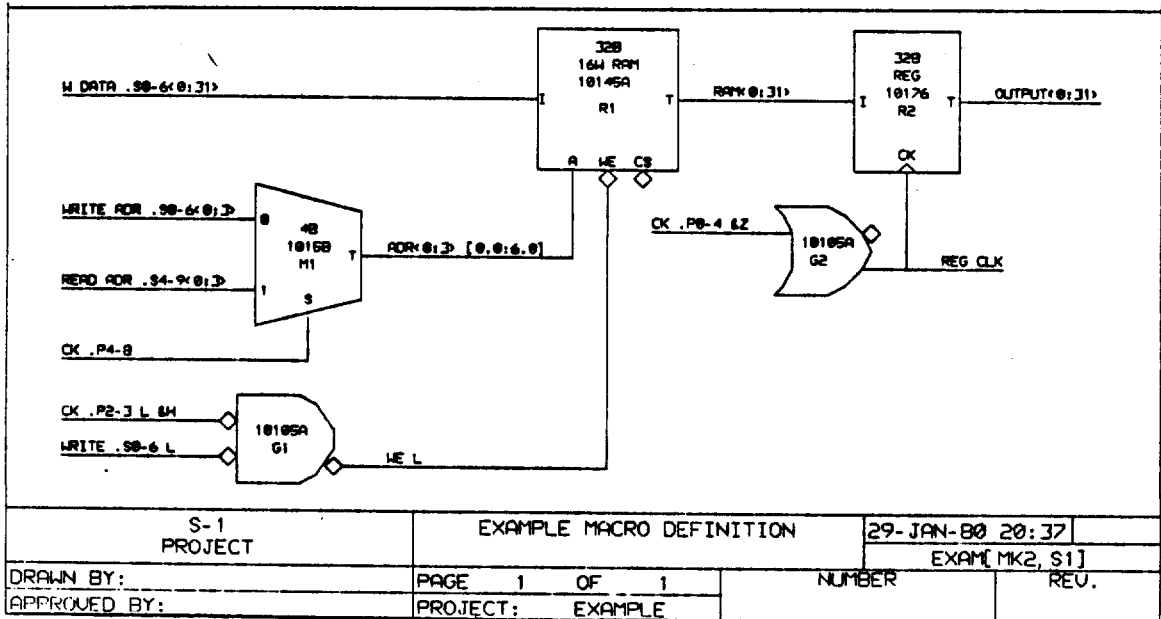


Figure 8
Example circuit to be verified.

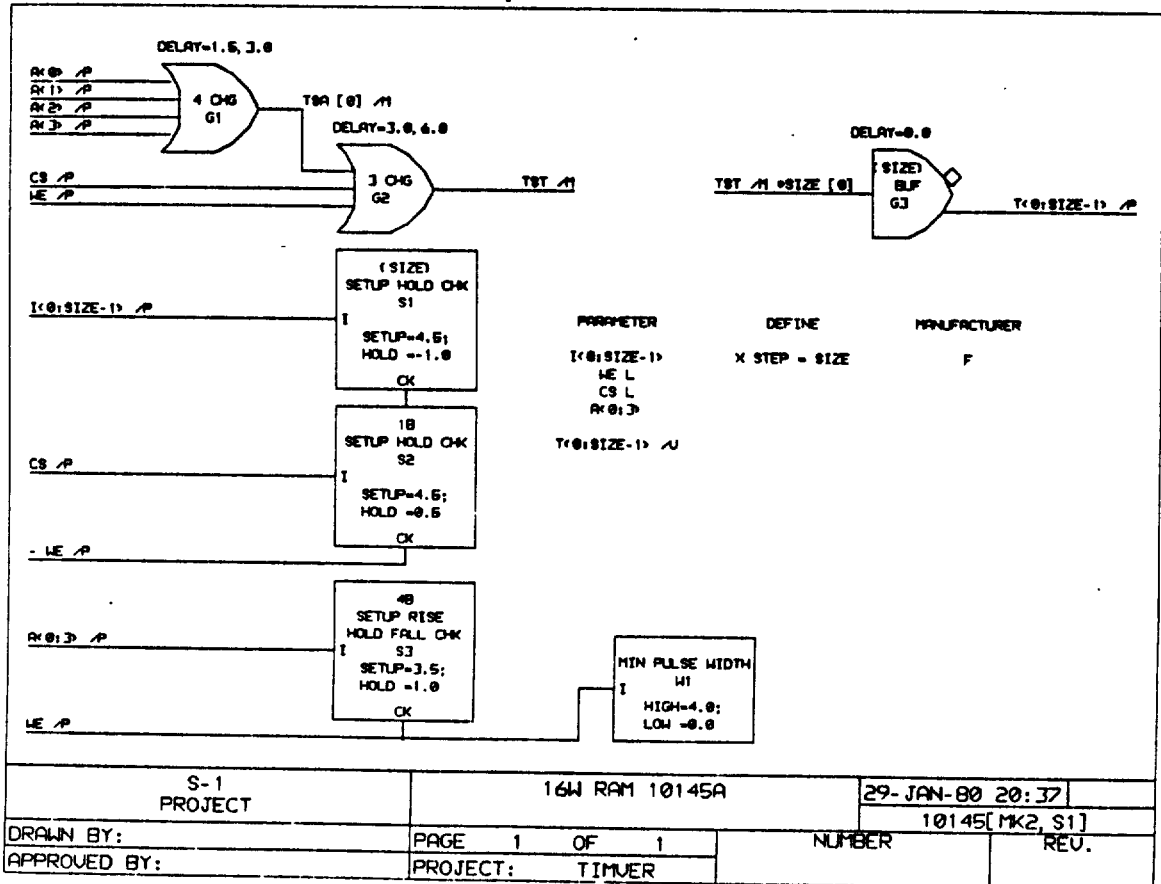


Figure 9
Definition of the 10145A, a 16-word random access memory chip (RAM).

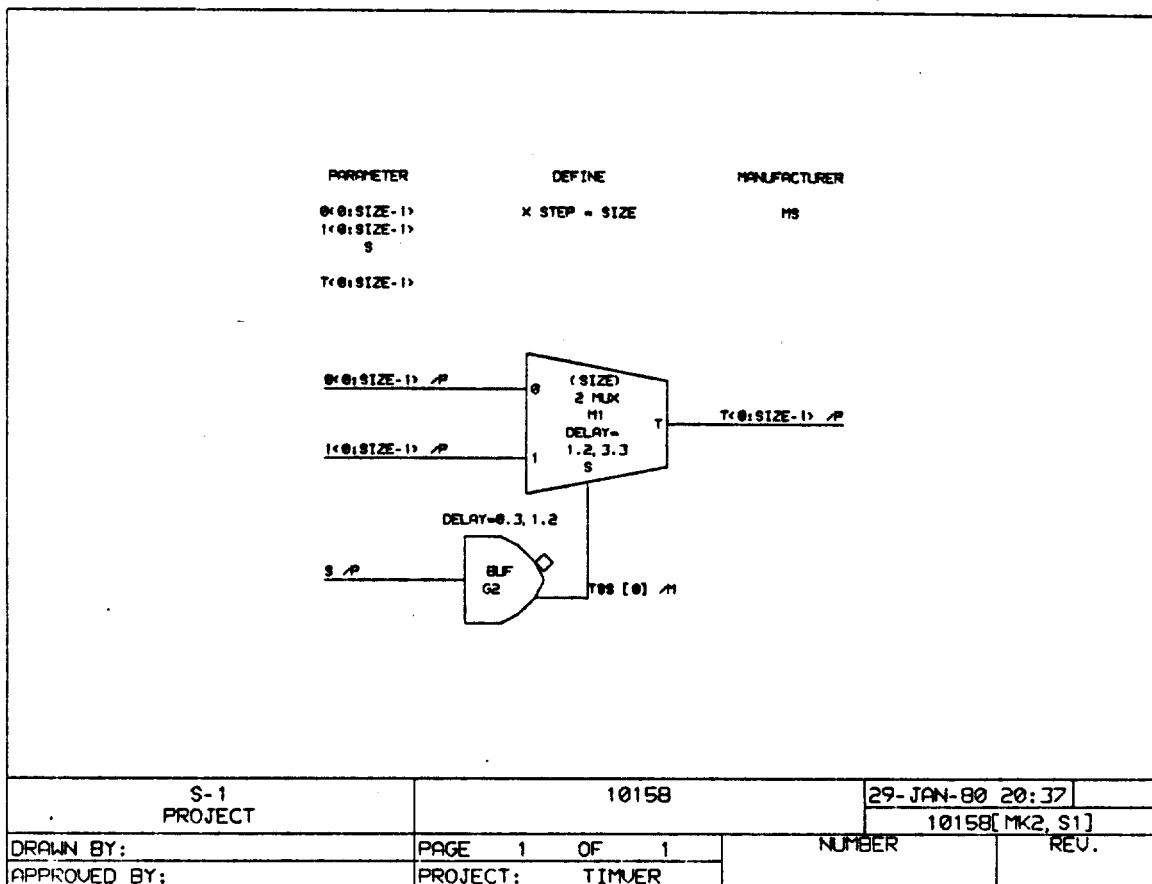


Figure 10
Definition of the 10158, a 2-input multiplexer chip.

Values of all signals

ADR<0:3>	S:0.0, C:0.5, S:5.5, C:25.5, S:30.5
CK .P0-4	R:0.0, I:1.0, F:24.0, 0:26.0, R:49.0 (constant value)
CK .P2-3	0:0.0, R:11.5, I:13.5, F:17.8, 0:19.8 (constant value)
CK .P4-8	P:0.0, 0:1.0, R:24.0, I:26.0, F:49.0 (constant value)
OUTPUT<0:31>	S:0.0, C:0.5, S:7.5
RANK<0:51>	S:0.0, C:5.0, S:20.5, C:30.0, S:45.5
READ ADR .S4-9<0:3>	S:0.0, C:6.3, S:25.0
REG CLK	R:0.0, I:1.0, F:24.0, 0:26.0, R:49.0
W DATA .S0-6<0:31>	S:0.0, C:37.5
WE	0:0.0, R:11.5, I:13.5, F:17.8, 0:19.8
WRITE .S0-6	S:0.0, C:37.5
WRITE ADR .S0-6<0:3>	S:0.0, C:37.5

Figure 11
Output from the Timing Verifier, showing values of signals.

Setup, Hold and Minimum Pulse Width errors

Setup time error: Setup Time = 3.5, Hold Time = 1.0	
CK INPUT = WE	(+0.0)
DATA INPUT = ADR	(+0.0)
	0:0.0, R:11.5, I:13.5, F:17.8, 0:21.8
	S:0.0, C:0.5, S:11.5, C:25.5, S:36.5
Setup time error: Setup Time = 2.5, Hold Time = 1.5	
CK INPUT = REG CLK	(+0.0)
DATA INPUT = RAM	(+0.0)
	R:0.0, I:3.0, F:24.0, 0:26.0, R:49.0
	S:0.0, C:5.0, S:22.5, C:30.0, S:47.5

Figure 12
Set-up and hold errors found by the Timing Verifier.

WIRE DELAY ESTIMATES AND CALCULATIONS

Before the actual wire delays are known, the Timing Verifier uses a set of rules to estimate them, except where they have been specified by the designer. After the actual routing of the wires and the simulation of their transmission line properties has been completed by the SCALD Physical Design Subsystem [3,4], accurate wire delays are available. These are then fed back into the Timing Verifier, which does a detailed check of the timing of the design.

CIRCUIT VERIFICATION EXAMPLE

Figures 11 and 12 show the results of running the circuit shown in Figure 8 through the Timing Verifier, with a specified cycle time for the circuit of 50 nsec, and a default wire delay of 0 to 2 nsec. Figure 11 gives a complete listing of all of the signals, showing their value as a function of time. Consider the first signal in the list, "ADR<0:3>". It has the same value for all of its bits, and so has only one value given. It is stable at the beginning of the cycle, and starts changing 0.5 nsec into the cycle. It is then changing from 0.5 nsec to 5.5 nsec, at which point it goes stable until 25.5 nsec. It is then changing from 25.5 nsec into the cycle until 30.5 nsec. It then goes stable from 30.5 nsec until the end of the cycle.

Figure 12 lists the set-up and hold time errors. Because of the long wire specified on the signal "ADR<0:3> [0.0&0.0]", two setup time errors are generated. The first error message shows the address on the RAM just going stable at time 11.5, the same time as the write enable (WE) signal starts rising. Since a RAM requires a setup time of 3.5 nsec, the wire delay on the address signal must be reduced to 2.5 nsec in order to eliminate the error. The second error message listed shows the data being read out of the RAM going stable at time 47.5 nsec, and the clock starting to rise at time 49.0 nsec, giving only 1.5 nsec of set-up time instead of the required 2.5 nsec.

CORRELATIONS WITHIN DIGITAL SYSTEMS

Consider constructing an 8-bit shift register using two 4-bit shift register chips. The shift output of one chip must be connected to the shift input of the other chip. If the minimum delay from the clock to the shift output is not greater than the hold time on the shift input by at least the maximum skew possible between the two clock inputs, then there is a timing problem. The key to verifying this timing constraint is to calculate the maximum skew between the two clock inputs, taking into account any correlations within the circuit.

Now consider the case where there is a large amount of skew on the time at which the clock signal will occur, but that the two clock inputs are wired together with a short wire. The skew that each chip sees is large, but the maximum possible skew between the two inputs is quite small, because of the correlation between the two clock inputs. To analyze this case correctly, the Verifier needs to understand the correlation between the two clocks. The approach that the Timing Verifier has taken is to make the designer explicitly declare this correlation in the design specification, relieving the system of the burden of automatically finding it. Since this type of correlation tends to occur in only a few simple SCALD macros (defining items such as shift registers and counters), it seems a small burden to have to declare when a circuit is depending on such a correlation to in order work properly.

VERIFICATION OF THE S-1 MARK IIA

The Timing Verifier has been used to check all of the timing constraints in the S-1 Mark IIA processor design. The design was checked in two parts, consisting of the instruction and operand preparation unit and the instruction execution unit. Each of these units consists of approximately 5,000 MSI and LSI ECL chips. The verification of one of these units takes about 15 to 20 minutes of execution time and uses 6 to 8 megabytes of storage on the S-1 Mark I processor. The Mark I processor is the first generation S-1 processor which was designed with SCALD, and is comparable in performance to an IBM 370/168.

The verification of the timing constraints proceeded on a daily basis during the S-1 Mark IIA design process. Each day the Timing Verifier was used to find any timing errors which might have been introduced into the design during that day's work. This procedure allowed errors to be found and fixed as they were introduced into the design, before their effects could propagate significantly throughout the design. This early, nearly continuous feedback was found to be invaluable in timely completion of a design with a high degree of confidence in its operability at full speed.

CONCLUSIONS

The SCALD Timing Verifier has been a very efficient way of discovering timing errors in the design of large digital systems. It is highly cost-effective from the standpoint that it requires little effort from the designer beyond what is required to execute the basic design. It is also computationally efficient, allowing a large design to be verified in a relatively small amount of computer time and memory.

Once the timing constraints have been verified, then a simple logic simulator, one which does not have to worry about any timing problems, can find the relatively probable logic errors. The improbable logic errors can then be found either by a hardware simulator or a prototype. The timing in both the prototype and the final production implementation can then be checked with the Timing Verifier.

ACKNOWLEDGMENTS

I would like to thank Forest Baskett and Bill vanCleave for the constant support and guidance they have provided throughout the course of this research. The Fannie and John Hertz Foundation's gracious support has provided me the freedom to pursue this research. Bill Bryson, Mike Farmwald, and Jeff Rubin have been the first users of the SCALD Timing Verifier and their patience and many suggestions for improvement are most appreciated. My thanks also go to the Office of Naval Research and the Naval Electronics System Command for the support to the S-1 Project which has provided the necessary environment for this research, and to Curt Widdoes and Lowell Wood, whose tireless efforts made the formation of the S-1 Project possible.

This work was in part performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore Laboratory under contract No. W-7405-ENG-48.

REFERENCES

1. Kusik, R. and Wesley, P. "Hierarchical Logic Simulation for Digital Systems Development," Proc. Electro/76, Boston, Mass., May 1976, pp. 26.3.1-26.3.8.
2. McWilliams, T.M. and Widdoes, L.C., "SCALD: Structured Computer-Aided Logic Design," Proceedings of the Fifteenth Design Automation Conference, Las Vegas, Nev., June 1978, 271-277.
3. McWilliams, T.M. and Widdoes, L.C., "The SCALD Physical Design Subsystem," Proceedings of the Fifteenth Design Automation Conference, Las Vegas, Nev., June 1978, 278-284.
4. S-1 Project Staff, "Advanced Digital Computing Technology Base Development for Navy Applications: The S-1 Project," Prepared for the Naval Systems Division, Office of Naval Research and the Research and Technology Directorate, Naval Electronics Systems Command, September 30, 1978. (UCID-18038)
5. Szygenda, S.A., "TEGAS2—Anatomy of a General Purpose Test Generation and Simulation System for Digital Logic," Proceedings of the ACM IEEE Design Automation Workshop, June, 1972, 116-127.
6. Wold, M.A., "Design Verification and Performance Analysis", Proceedings of the Fifteenth Design Automation Conference, Las Vegas, Nev., June 1978, 264-270.

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Department of Energy, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately-owned rights.

Reference to a company or product name does not imply approval or recommendation of the product by the University of California or the U.S. Department of Energy to the exclusion of others that may be suitable.